INRIA

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France

Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 375

ERROR DIAGNOSIS IN LOGIC PROGRAMMING, AN ADAPTATION OF E. Y. SHAPIRO'S METHOD

Gérard FERRAND

Mars 1985

Error diagnosis in Logic Programming, an adaptation of E.Y. SHAPIRO's method.

By Gérard FERRAND

INRIA
Domaine de Voluceau - Rocquencourt
BP 105
F-78153 Le Chesnay cédex

Département Mathématiques-Informatique
Université d'Orléans
F-45046 Orléans Cédex

Abstract :

In the framework of Horn Clauses programming "fixpoint", non-procedural semantics, we study and extend the Shapiro's method of logic program bugs diagnosis, by a logic program. For that purpose we must define a "relative denotation" of logic programs. We follow Clark's approach to logic program semantics, which is more general than the most usual approach. We also give a new formulation of these already known approaches in a unified framework.

Note : This work has been supported by the "GRECO de programmation" (ATTRISEM project).

Résumé :

Dans le cadre de la sémantique "point fixe", non procédurale de la Programmation en Clauses de Horn, on étudie et étend la méthode de Shapiro de détection, à l'aide d'un programme logique, d'erreurs dans des programmes logiques. On doit pour cela définir une "dénotation relative" pour les programmes logiques. On suit l'approche de Clark de la sémantique des programmes logiques, plus générale que l'approche la plus usuelle. On donne aussi une nouvelle formulation, dans un cadre unifié, de ces approches connues.

Nota : Ce travail a été réalisé dans le cadre du "GRECO de programmation" (projet ATTRISEM).

Contents

1 - Introduction

The starting point of this paper is the article "Algorithmic Program Debugging" by E.Y. Shapiro [8]. In short, the motivation is the study of a system which partially mechanizes the diagnosis of bugs in a program. This bug diagnosis consists in a dialogue between the system and the programmer : the program which has been actually written by the programmer is a data for the system, the programmer has to answer questions on the semantics of the program he would have liked to write.

For example, if the programmer has seen that the execution of his program ends by giving a wrong result, he uses the system to spot a bug in his program, but, as it may be a non-deterministic program, he can also use the system if he has seen that the execution ends without giving any result. (In his thesis [9], Shapiro has developped his system and even examines the problem of non-ending, and optimisation questions as well).

More precisely, Shapiro defines appropriate notions on the correctness of programs for a certain class of languages, and a notion of "oracle" which can be asked questions about the semantics of a program, then he describes algorithms with which one can diagnose incorrectness, which he so defined, in programs of this class.

Then he implements these algorithms as PROLOG programs (Edinburgh Prolog-10) which are themselves the system, that is used to bug diagnosis
in programs in this PROLOG.

PROLOG is choosen by Shapiro as the most appropriate tool, but the justifications are given in a framework of general programming.

In this paper, we shall examine what this theoretical framework becomes, and we intend to develop it, when one chooses to remain in the field of logic programming - and more precisely in Horn clauses programming [4]. There is a natural motivation for this process : because of its "logic" origin, it is a privileged area to study what concerns errors. But more precisely we have the opportunity of semantics [1,2,3]

which is at the same time of logical essence (notion of model) and of fixpoint type (in the sense of Scott) and which is rather simple (manipulations of sets of atomic formulas). One can then consider treating correctness, in the framework of this semantics, not only of the "erroneous programs" but also of the "bug diagnoser programs", precising what they can compute, and so to reinforce and extend in this particular framework Shapiro's definitions and justifications.

We would like to insist on the fact that we choose the framework of Horn clauses logic programming, and we shall talk about the continuation of this paper in the conclusion only - continuation consisting in examining a PROLOG implementation. In what follows, the references to PROLOG are used to give motivations, or are remarks about Shapiro's system.

More precisely, here in this framework we shall study two "dual" problems defined by Shapiro : incorrectness diagnosis and insufficiency diagnosis. We then define the "symptoms" of incorrectness and of insufficiency which, when they are seen, induce the programmer to look for his errors (incorrectness or insufficiency), and we can precisely describe the link between symptom and found out error.

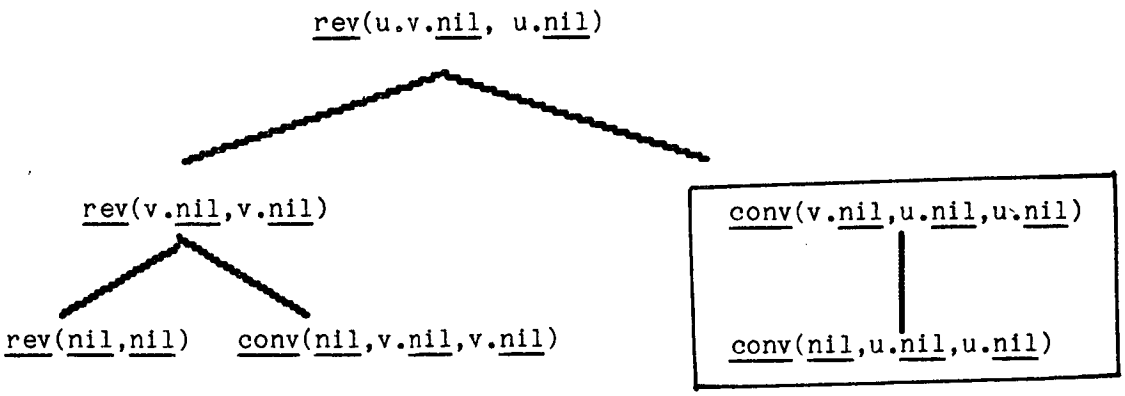
Here are two examples simple enough to be presented without waiting for the general definitions. In these two examples the "expected" semantics for the predicate rev(x,y) is reversing a list, for conc(x,y) it is concatenation : with usual notations the empty list is represented by nil, and the list $(a_1\ a_2\ \ldots\ a_n)$ by $.(a_1, .(a_2,\ldots.(a_n,nil)\ldots))$ in practice noted $a_1 . a_2 \ldots . a_n . nil$. By using the constants 1 and 2, rev(1.2. nil, 2.1. nil) and conc(1.2 nil, 3. nil, 1.2.3. nil) are examples of atomic formulas of this "expected" semantics, but to illustrate another question studied here at the same time and presented a little further down in this introduction, we would rather use atomic formula with variables (x,y,z,t,u,v,w,...).

1st example : let P_1 be the logic program made of the 4 following clauses

rev(nil, nil) ←
rev(x.y,z) ← rev(y,t), conc(t,x.nil,z)
conc(nil,x,x) ←
conc(x.y,z,t) ← conc(y,z,t)

In fact, in the actual semantics of P_1 there is the "erroneous" atomic formula
rev(u.v.nil,u.nil), that we shall call "incorrectness symptom". We may consider that the "cause of this symptom" is the instance
conv(v.nil, u.nil, u.nil) ← conc(nil, u.nil, u.nil) of the last clause of P_1, instance we shall call incorrectness : in an incorrectness, the conclusion is "erroneous" while the premisses are not. The link between incorrectness symptom and incorrectness is described formally by a tree as the following :

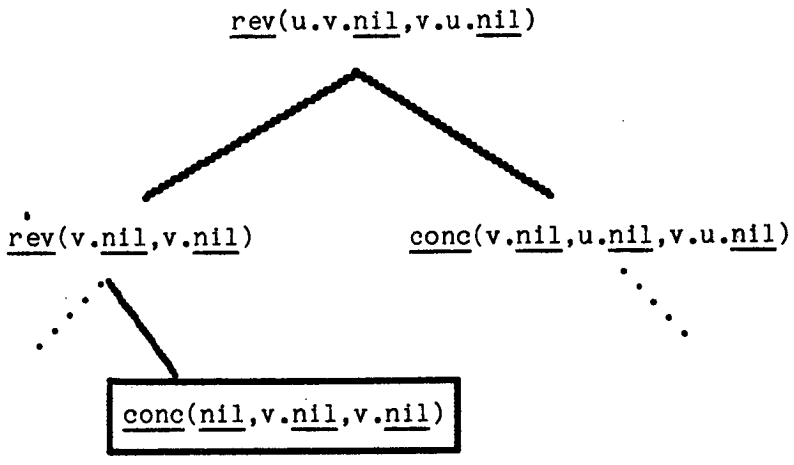
(The symptom is at the root, we have drawn a frame around the incorrectness).



2 nd example : let P_2 be the logic program made of the 3 following clauses :

rev(nil,nil) ←
rev(x.y,z) ← rev(y,t), conc(t,x.nil,z)
conc(x.y,z,x.t) ← conc(y,z,t)

Now in the actual semantics of P_2 there is no atomic formula of the form rev(u.v.nil,w) while one "expected" rev(u.v.nil, v.u.nil), that we shall call "insufficiency symptom". We may consider that "the cause of this symptom" is conc(nil, v.nil, v.nil), we shall call insufficiency :
an insufficiency is an atomic formula which belongs to the "expected" semantics but is not conclusion of an instance of a clause whose premisses would be in the "expected" semantics. The link between insufficiency symptom and insufficiency is described formally by a tree as the following (an insufficency is always a leaf) :



A logic program "incorrectness diagnoser", applied to P_1, will have itself a semantics which will have an atomic formula which associates the incorrectness to the symptom, for example a formula of the form :
incorrectness(rev(u.v.nil,u.nil), conc(v.nil,u.nil,u.nil) if conc(nil,u.nil,u.nil))
(where the if has a formal role of function symbol, and rev and conc too).

In the same way a logic program "insufficiency diagnoser", applied to P_2, will have an atomic formula in its semantics of the form, for example:

insufficiency(rev(u.v.nil,v.u.nil), conc(nil,v.nil,v.nil)).

But in order to undertake this study we have to extend the framework of the semantics we are speaking of, if only to give a meaning to the fact that the semantics of a logic program "bug diagnoser", fixed, depends on the logic program P_1 to which it is applied.

A remark on Shapiro's system written in PROLOG can help motivate this extension : this system accedes to the clauses of the given "erroneous" program thanks to predicates of clauses manipulations. Moreover, this system interacts with the programmer, thanks to input-output predicates. Not to forget that the "erroneous" program itself may have arithmetical calculus predicates for example (supposed to be errorless). These are three kinds of (PROLOG) predicates which cannot -by nature- be interpreted by the usual non-procedural way.

Let's go back now to logic programming. At first, to avoid confusions while respecting custom, we shall speak about denotation to name the semantics of a logic program, so that we can already distinguish the actual denotation of the erroneous program, from the ideal denotation expected by the programmer.

But so we are trying to extend this notion so that it can in particular apply in a positive way to the logic program "bug diagnoser" itself. However it is an extension of the field of application of the methods of this fixpoint, non-procedural semantics, which is not exclusively bound to the context of bug diagnosis.

For this extension we introduce the notion of relative denotation; this idea comes from the notion of relative recursivity and oracle machine [7] as well as from the notion of logical consequence.

It is obvious that the program "bug diagnoser" is not on the same level of language as the various erroneous programs to which it applies. This creates problems which we solve by using a "representation" method.

We also extend to another sense this usual semantics of the logic programs, thus following Clark [2], and we try at the same time to extend the results on bug diagnosis. The usual semantics [1,3] only uses closed atomic formulas, because of this one loses some of the information potentially contained in a logic program : to have a quite simple example, let us suppose that the only closed terms are the two constants 1 and 2. If the "question asked" is the goal R(x), this semantics cannot render the difference between the two cases of "answer" :

- the interpreter outputs the 2 substitutions

  x = 1, x = 2

- the interpreter outputs x = x.

In a general way one can argue about the double logical role played by the usual "minimal model" [1,3] of a logic program. On the one hand it identifies itself to a model of a certain set of axioms, which is the logic program, and then the elements of the domain of this model are the closed terms (Herbrand's universe). On the other hand the "minimal model" is also a set of closed atomic formulas : the ones which are true in the model (in the meaning used above), and which are the same as the ones which are a logical consequence of the pre-quoted axioms. But this remarkable property is not true any longer for atomic formulas with variables : the set of those which are true in this usual model do not coincide any longer with the set of those which are the logical consequences of the pre-quoted axioms. Nevertheless, it is natural to take into consideration all the atomic formulas, non-necessarily closed, since the 'answers" of the interpreter are non-necessarily closed substitutions.

Like Clark [2], we use all the atomic formulas in the denotation of a logic program, and not only the closed ones, and so we defined its general denotation. Then we emphasize the relation between the most usual (closed) denotation and the "finitely generated" models, as well as the double role played by the general denotation, both model and set of consequences. Moreover this extension of the usual semantics does not alter anything as concerns the closed atomic formulas : on them the two denotations (closed and general) coincide.

However, referring to the general denotation in bug diagnosis problem, one comes across a result of "limitation" (in a sense of incompleteness) intrinsic in the problem and independant from the method (of Shapiro), while this limitation disappears if one limits the problem to the closed denotation.

This report is organised as follows : In section 2, we gather and re-examine certain results on the fixpoint, non-procedural semantics, in a unified framework according to the two approaches of Apt, Van Emden, Kowalski [1,3] and Clark [2], to finally reach the relative, general denotation.

In section 3, we formulate the problems of incorrectness and insufficiency, following Shapiro. In section 4, we set some technical preliminaries. In sections 5 and 6, we respectively study incorrectness and insufficiency problems. In section 7, as a conclusion, we allude to the perspective of an application to a PROLOG implementation, which is the topic of a work-to-come.

2 - Denotations of a logic program

2.1. Syntax

It is the usual syntax of logic programming. But as the roles of term and of atomic formula will be relative to a "level", we are giving here some precise definitions and notations which we are going to use.

In all that follows, one considers a fixed, denumerable infinite set $\mathcal{V}$ called set of variables. The elements of $\mathcal{V}$ will always be noted x,y,z,...,u,v,...

A function symbols system $\mathcal{F}$ is defined by giving a set of symbols (disjointed from $\mathcal{V}$), each symbol having a certain arity. The function symbols with arity o are the constants of $\mathcal{F}$.

One defines in a usual way the set of terms built from $\mathcal{V}$ and $\mathcal{F}$ and one notes it down TERM ($\mathcal{F}$), $\mathcal{V}$ being fixed and implicit : each term is either a variable or is written in a unique manner $f(t_1,...,t_n)$ where f is a function symbol with arity n and $t_1,..,t_n$ are terms.

A predicate symbols system $\mathcal{P}$ is defined too by giving a set of symbols (disjointed from $\mathcal{V}$ and from that of $\mathcal{F}$), each symbol having an arity too. One defines in a usual way the set of atomic formulas, or atoms built from $\mathcal{V},\mathcal{F}$ and $\mathcal{P}$ and one notes it down ATOM($\mathcal{P}$, $\mathcal{F}$) : each atom is written in a unique manner $R(t_1,...,t_m)$ where R is a predicate symbol with arity m and $t_1,...,t_m$ are terms.

In all that follows, what is called clause coincides with the usual notion of definite and Horn clause : a clause is defined by giving an atom A and a finite set of atoms $\mathcal{C}$. It is then noted $A \longleftarrow \mathcal{C}$, or $A \longleftarrow B_1,...,B_k$ too if $\mathcal{C} = \{B_1,...,B_k\}$, and $A \longleftarrow$ if $\mathcal{C} = \emptyset$.

A substitution is a map $\theta : \mathcal{V} \longrightarrow$ TERM($\mathcal{F}$). If e is a term, atom, set of atoms, clause... one defines in a usual way the instance of e by θ, noted $e\theta$, result given by replacing in e each variable x by $\theta(x)$.

For a term, atom,..., closed means here without variable (then $e\theta = \theta$). One notes TERMC($\mathcal{F}$) and ATOMC ($\mathcal{P}$, $\mathcal{F}$) the set of closed terms and the set of closed atoms.

A logic program is a set of clauses.

2.2. ℓ-models

Here we would just like to precise usual notions of logic adapted to this syntax and to our goals.

An ℓ-interpretation M (for $\mathcal{P}$ and $\mathcal{F}$) is defined by giving :

a) A non-empty set $|M|$ (the domain of M).
b) For each function symbol f with arity n, a map noted $f_M : |M|^n \longrightarrow |M|$, and in particular, if n=o, an element of $|M|$.
c) For each predicate symbol R with arity m, a relation $R_M \subseteq |M|^m$.

An assignment in M is a map $s : \mathcal{V} \longrightarrow |M|$. The value in M, defined by s, of a term t is then defined in a usual way. It is an element of $|M|$, noted t^s and, if t is closed, noted t^M too because then t^s does not depend on s.

The ternary relation $M,s \models A$ is defined by writing :

$$M,s \models R(t_1,\ldots,t_m) \iff (t_1^s,\ldots,t_m^s) \in R_M$$

where $A=R(t_1,\ldots,t_m)$ is any atom.

It is defined for any clause $A \longleftarrow \mathcal{C}$ too in the following way :

$M,s \models A \longleftarrow \mathcal{C}$ if and only if the following implication is true :

$$(M,s \models B \text{ for each } B \in \mathcal{C}) \Longrightarrow M,s \models A.$$

The clause $A \longleftarrow \mathcal{C}$ is valid in M, property noted $M \models A \longleftarrow \mathcal{C}$, if $M, s \models A \longleftarrow \mathcal{C}$ for each s.

M is an ℓ-model of the logic program P, property noted $M \models P$, if each clause in P is valid in M.

$A \longleftarrow \mathcal{C}$ is consequence of P, property noted $P \vdash A \longleftarrow \mathcal{C}$, if $A \longleftarrow \mathcal{C}$ is valid in each $\mathcal{L}$-model of P.

The validity and consequence notions also apply to the atoms by replacing the atom A by the clause $A \longleftarrow$.

Let us now introduce some definitions which will allow us to give a new formulation of the approaches of Apt, Van Emdem, Kowalski [1,3] and of Clark [2] .

Definition 2.1.

A termal domain H is a non empty subset of TERM($\mathcal{F}$) such as $t\theta \in H$ for each term $t \in$ TERM($\mathcal{F}$) and each substitution θ with range in H ($\theta(x) \in H$ for each $x \in \mathcal{U}$). The termal base BASE(H) associated to H is the set of the atoms $R(t_1,\ldots,t_m)$ where $t_i \in H$.

TERM($\mathcal{F}$) (usual, or closed, Herbrand's universe [1,3]) and TERMC($\mathcal{F}$) (that we may call general Herbrand's universe) are two termal domains, and we necessarily have TERMC($\mathcal{F}$) $\subseteq$ H $\subseteq$ TERM($\mathcal{F}$) (For H = TERM($\mathcal{F}$), BASE(H) is the usual Herbrand's base).

Definition 2.2.

An $\mathcal{L}$-interpretation M (for $\mathcal{P}$ and $\mathcal{F}$) is said to be termal, on the termal domain H, if

a) $|M| = H$

b) each f_M is the canonical map

$$f_M(t_1,\ldots,t_n) = f(t_1,\ldots t_n)$$

By definition $t_1,\ldots,t_n \in H \Longrightarrow f(t_1,\ldots,t_n) \in H$, so for a given H, defining a $\mathcal{L}$-interpretation M termal on H is simply defining the relations R_M on H. Then, as maps, we can identify the assignments in M, with the substitutions with range in H, but moreover we have $t^{\theta} = t\theta$ for any term $t \in$TERM($\mathcal{F}$) and any assignment θ, and so $M,\theta \models R(t_1,\ldots,t_m) \Longleftrightarrow (t_1\theta,\ldots,t_m\theta) \in R_M$.

As Clark [2] we are trying to characterize the notion of consequence with the only help of termal models, but we use a slightly different method :

Notation 2.3.

H being a fixed termal domain, M any ℓ-interpretation and s an assignment in M, we note M(H,s) the ℓ-interpretation, termal on H, defined by $R_{M(H,s)} = \{(t_1,\ldots,t_m) \in H^m \mid M,s \models R(t_1,\ldots,t_m)\}$ for each predicate symbol R with arity m.

Then we can easily have :

lemma 2.4.

For any clause $A \longleftarrow \mathcal{C}$ and any assignment θ in M(H,s),
$M(H,s), \theta \models A \longleftarrow \mathcal{C} \iff M,s \models A\theta \longleftarrow \mathcal{C}\theta$

Lemma 2.5. For any logic program P

$M \models P \Longrightarrow M(H,s) \models P$

Proof : from lemma 2.4. and thanks to the implication

$$M \models A \longleftarrow \mathcal{C} \Longrightarrow M \models A\theta \longleftarrow \mathcal{C}\theta$$

Lemma 2.6 : In each of the two following cases :

a) $H = TERM(\mathcal{F})$ (and $A \longleftarrow \mathcal{C}$ any clause)
b) $A \longleftarrow \mathcal{C}$ closed (and H any termal model)
we have :
$M(H,s) \models A \longleftarrow \mathcal{C} \Longrightarrow M,s \models A \longleftarrow \mathcal{C}$

Proof : from lemma 2.4., with $\theta(x)=x$ in case a) and any θ in case b).

Definition 2.7.

The clause $A \longleftarrow \mathcal{C}$ is consequence on H of P, property noted $P \overset{H}{\vdash\!\!\vdash} A \longleftarrow \mathcal{C}$, if $A \longleftarrow \mathcal{C}$ is valid in each termal on H, $\mathfrak{L}$-model of P. (The definition applies to an atom A by replacing it by the clause $A \longleftarrow$).

Theorem 2.8.

If $H=TERM(\mathcal{F})$, or if $A \longleftarrow \mathcal{C}$ is closed, we have :

$$P \vdash A \longleftarrow \mathcal{C} \iff P \overset{H}{\vdash\!\!\vdash} A \longleftarrow \mathcal{C}$$

proof : use lemmas 2.5 and 2.6

In particular we have, for $H = TERM(\mathcal{F})$ and any atom A, $P \vdash A \Longleftrightarrow P \overset{H}{\vdash\!\!\vdash} A$, result which is proved in Clark [2] and which shows that, as concerns atomic consequences of P, it is sufficient to consider the $\mathfrak{L}$-models of P which are termal on H.

In general, $P \vdash A \longleftarrow \mathcal{C}$ is not equivalent to $P \overset{H}{\vdash\!\!\vdash} A \longleftarrow \mathcal{C}$. Let us say that an $\mathfrak{L}$-interpretation M is finitely generated if each element of $|M|$ is the value of a closed term. Then we have in particular :

Proposition 2.9.

If $H = TERMC(\mathcal{F})$ and $A \longleftarrow \mathcal{C}$ is any clause, $P \overset{H}{\vdash\!\!\vdash} A \longleftarrow \mathcal{C}$ if and only if $A \longleftarrow \mathcal{C}$ is valid in all the finitely generated $\mathfrak{L}$-models of P.

proof : Use the following remark, with lemmas 2.4 and 2.5. :

If M is finitely generated, for each assignment s in M there exists a substitution θ with range in H such that, for each term t, $t^s = t\theta^M$, and so $M,s \models A \leftarrow \mathcal{C} \iff M \models A\theta \leftarrow \mathcal{C}\theta$ ($\iff M,s' \models A\theta \leftarrow \mathcal{C}\theta$ for any s' because θ is closed).

2.3. Models.

The following definitions are introduced in order to only use subsets of BASE(H) and in particular to get the notions of model of Apt, Van Emden, Kowalski [1,3] when H = TERMC($\mathcal{F}$), and of Clark [2] when H = TERM($\mathcal{F}$).

Definition 2.10.

An interpretation on H is any subset of BASE(H).

There is a one-to-one and onto correspondence between the interpretations on H and the ℓ-interpretations which are termal on H : to the ℓ-interpretation M, termal on H, we associate the interpretation I on H defined by :

$$I = \{R(t_1,\ldots,t_m) \in \text{BASE(H)} \mid (t_1,\ldots,t_m) \in R_M\}$$

(We may remark that $(t_1,\ldots,t_m) \in R_M$ is equivalent to $M,\theta \models R(x_1,\ldots,x_m)$ where $\theta(x_i)=t_i$, but in general is not equivalent to $M \models R(t_1,\ldots,t_m)$).

Definition 2.11.

The validity on H of a clause (and an atom), in an interpretation on H, is the notion associated, by this correspondence, to the validity in an ℓ-interpretation (termal on H).

In the same way, a model on H of the logic program P is an interpretation on H associated to an ℓ-model (termal on H) of P by this correspondence.

Then we have :

Proposition 2.12.

Let I be an interpretation on H, M the associated $\mathcal{L}$-interpretation (termal on H), and θ a substitution with range in H (which also is an assignment in M). Then for each atom A we have :
$M,\theta \models A \Longleftrightarrow A\theta \in I$

And it follows that

Proposition 2.13.

Let I be an interpretation on H. The clause $A \longleftarrow \mathcal{C}$ is valid on H in I if and only if, for each substitution θ with range in H, $\mathcal{C}\theta \subseteq I \Longrightarrow A\theta \in I$. The atom A is valid on H, in H, if and only if, for each substitution θ with range in H, $A\theta \in I$.

we get usual definitions and results [1] by posing :

Notations 2.14.

$INST_H(P)$ = the set of the clauses which are instances of a clause in P by a substitution with range in H.

T_P^H = the map, from the set of the interpretations on H, in itself, defined by :

$$T_P^H(I) = \{A \in BASE(H) \mid \text{there exists } \mathcal{C} \subseteq I \text{ such that } A \longleftarrow \mathcal{C} \in INST_H(P)\}$$

$MOD_H(P)$ = the set of models on H of P.

We easily get the following generalization of usual results :

Proposition 2.15.

$I \in MOD_H(P) \iff T_P^H(I) \subseteq I$
$I \subseteq J \Rightarrow T_P^H(I) \subseteq T_P^H(J)$
$\bigcap MOD_H(P) \in MOD_H(P)$

Then $\bigcap MOD_H(P)$ is the least model on H of P.

Moreover,

Lemme 2.16.

Let A be any atom. Then A is valid on H in $\bigcap MOD_H(P)$ if and only if $P \vdash^H A$

Proof : A valid on H in $\bigcap MOD_H(P)$
$\iff$ for each θ with range in H, $A\theta \in \bigcap MOD_H(P)$
$\iff$ for each θ...for each $I \in MOD_H(P)$ $A\theta \in I$
$\iff$ for each $I \in MOD_H(P)$ for each θ... $A\theta \in I$
$\iff$ for each $I \in MOD_H(P)$ A valid in I
$\iff P \vdash^H A$.

2.4. The denotations of a logic program :

The usual semantics of a logic program [1,3] corresponds to the following definition :

Definition 2.17.

The closed denotation of the logic program P, noted CD(P). is the least model on TERMC($\mathcal{F}$) of P, that is to say : $CD(P) = \bigcap MOD_H(P)$ with H = TERMC($\mathcal{F}$).

Then we get a usual result [3] according to which, for closed atoms, this only model characterizes the notion of consequence (in which intervene all the $\mathfrak{L}$-models, even those non termal) :

Proposition 2.18.

CD(P) is the set of closed atomic consequences of P, that is to say $CD(P) = \{A \in ATOMC(\mathcal{P},\mathcal{F}) \mid P \vdash A\}$.

Proof : for A closed, A valid in I $\Longleftrightarrow$ $A \in I$ (proposition 2.13), and use lemma 2.16 and theorem 2.8.

But this closed denotation is not sufficient to express that any atom is consequence of P. More precisely the set of the atoms valid in CD(P) that is to say the atoms whose each closed instance is in CD(P), in general is greater than the set of the atoms which are consequences of P. This comes from the following result :

Proposition 2.19 :

Let A be any atom. All the closed instances of A are in CD(P) if and only if A is valid in all the finitely generated $\mathfrak{L}$-models of P.

proof : from lemma 2.16 and proposition 2.9.

From a non-procedural point of view, inspired by Clark [2] one can figuratively say that a "question" Q is a set of atoms, and a substitution θ is an "answer", given by the program P to the question Q, if each atom of the set $Q\theta$ is a consequence of P. Clark proves the equivalence of such a semantics and of the well known procedural semantics, based on the resolution (Padawitz [5] gets a similar result in an algebric framework).

The following definitions corresponds to the non-procedural semantics of Clark [2]. In all that follows, if we do not precise H any longer, as concerns models, T_P,... it will be implicit that $H = TERM(\mathcal{F})$

Definition 2.20.

The (general) denotation of P, noted D(P), is the least model of P, that is to say : $D(P) = \bigcap MOD(P)$.

Clark proves that for any atom A, A is consequence of P if and only if A is valid in D(P), that is to say that each instance (closed or not) of A is in D(P). Here this property immediately comes from lemma 2.16 and theorem 2.8, and it already shows that the only model D(P) charaterizes the notion of consequence, for any atom. But D(P) satisfies a stronger property, which simplifies the situation. More precisely let us compare the two following properties, where I is a model of P and A is any atom :

a) $A \in I$

b) A valid in I, that is to say (proposition 2.13) all the instances of A are in I.

Obviously b) implies a) but in general the converse is false : we have a simple counter-example with

$P = \{p(1) \leftarrow q(1),\ q(1) \leftarrow,\ r(2) \leftarrow\}$

$I = \{p(1),\ q(1),\ r(2),\ q(x)\}$, $A = q(x)$, because then $q(2) \notin I$.

But we shall see that, in the particular case where $I = D(P)$, we have the equivalence between a) and b), so that the condition "A consequence of P" is not only equivalent to "each instance of A is in D(P)" but merely to "$A \in D(P)$".

Things seem involved only because the same formal object D(P) plays two roles at once, which have different logical natures : the role of a model of P, and the role of a theory generated by P, that is to say the set of the consequences (only the atomic ones) of P.

Theorem 2.21.

D(P) is the set of atomic consequences of P, that is to say
$D(P) = \{A \in ATOM(\mathcal{P}, \mathcal{F}) \mid P \vdash A\}$

Proof : let $C(P) = \{A \in ATOM(\mathcal{P}, \mathcal{F}) \mid P \vdash A\}$

a) For any clause $A \longleftarrow \mathcal{C}$ in P and any substitution θ we easily get $\mathcal{C}\theta \subseteq C(P) \Longrightarrow A\theta \in C(P)$ that is to say that $A \longleftarrow \mathcal{C}$ is valid in C(P). Thus C(P) is a model of P, and thus $D(P) \subseteq C(P)$.

b) By theorem 2.8 and lemma 2.16, if $A \in C(P)$ then A is valid in D(P), and in particular $A \in D(P)$. Thus $C(P) \subseteq D(P)$.

The following remarks can be justified in another way, but they immediately follow from this characterization of D(P) with the notion of consequence :

a - $CD(P) = D(P) \cap ATOMC(\mathcal{P}, \mathcal{F})$

b - For any atom A in $ATOM(\mathcal{P}, \mathcal{F})$, the fact that $A \in D(P)$ is not altered if we add new symbols to $\mathcal{F}$ and $\mathcal{P}$ (without altering P, but in general this alters D(P)).

In order to give an intuitive motivation for the following definition, let us quit Logic Programming as such, and let us imagine a PROLOG program with an arithmetical predicate R(x,y,z), and a set $\mathcal{E}$ in which there are the true instances of that predicate. Otherwise let us imagine that the computation of R(x,y,z) calls outer data, for example to instanciate x,y,z, thanks to a reading procedure, and that in $\mathcal{E}$ are the corresponding instances of R(x,y,z). Then to some extent it is similar to relative recursivity and oracle machines [7].

Definition 2.22.

$\mathcal{E}$ being a set of atoms, let us note (for form's sake) $\overleftarrow{\mathcal{E}}$ the set of clauses $\{A \longleftarrow \mid A \in \mathcal{E}\}$.

The (general) relative to $\mathcal{E}$, denotation of P noted $D(P \mid \mathcal{E})$ is the (general) denotation $D(P \cup \overleftarrow{\mathcal{E}})$ of the logic program $P \cup \overleftarrow{\mathcal{E}}$.

Obviously by $D(P \mid \emptyset) = D(P)$ which we may call absolute denotation to avoid confusion, and more generally we easily have
$D(P \mid \mathcal{E}) = D(P) \iff \mathcal{E} \subseteq D(P)$.

2.5 Fixed points.

Here we recall usual results, which come from Knaster-Tarski theorem [1] and give a "fixpoint semantics" to logic programs using the map T_P (notation 2.14).

Not only D(P) is the least model of P ($T_P(D(P)) \subseteq D(P)$) but also D(P) is the least fixed point of T_P ($T_P(D(P)) = D(P)$). In a dual way, there is a greatest fixed point of T_P, we shall note gfp (T_P), which is the greatest I such as $I \subseteq T_P(I)$ too.

This is only because T_P is an increasing monotonous map. Moreover, because in each clause $A \longleftarrow \mathcal{C}$ the set $\mathcal{C}$ is finite, one also has $D(P) = \bigcup_{n \in \mathbb{N}} T_P^n(\emptyset)$.

But the dual, which would be
gfp (T_P) = $\bigcap_{n \in \mathbb{N}} T_P^n(\mathrm{ATOM}(\mathcal{P}, \mathcal{F}))$, in general is not true [1].

3. The two problems we are going to study

3.1 Correctness and sufficiency

The following definitions are natural adjustments to logic programming of usual notions :

Definition 3.1.

Let P be a logic program and I an interpretation.

P is totally correct in I if $D(P) = I$
P is partially correct in I if $D(P) \subseteq I$
P is complete for I if $I \subseteq D(P)$

But these questions do not arise for any interpretation I (merely a tool to define consequence notion) but for an I liable to be the denotation of a logic program. Therefore we introduce the following definition (needless if we would only consider closed denotation).

Definition 3.2.

An interpretation I is an expected denotation if it is closed by substitution, that is to say that each instance of an atom in I is an atom in I too.

Shapiro's method [8] to diagnose bugs is based on correctness and completness notions different from the previous ones. For logic programming they refer to closed denotation, but in the same way we apply them to general denotation in the following definition :

Definition 3.3

P is correct in I if I is a model of P, that is to say $T_P(I) \subseteq I$.
P is sufficient for I if $I \subseteq T_P(I)$.

The following remark is obvious [8].

P is correct in I and sufficient for I in the same time if and only if $T_P(I) = I$, which is a property weaker than $I = D(P)$ = least fixed point of T_P, equivalent to P totally correct in I, that is to say P partially correct in I and sufficient for I in the same time.

More Precisely :

a) P correct in I $\Longrightarrow$ P partially correct in I (because D(P) is the least model). But the converse is false (counter-example: $P = \{A \leftarrow B\}$, $I = \{B\}$ so $D(P) = \emptyset \subseteq I$).

b) Sufficiency and completeness are independent from each other. (Completeness without sufficiency : with $P = \{A \longleftarrow B,\ B \longleftarrow\}$, $I = \{A\}$, $D(P) = \{A,B\}$); (sufficiency without completeness : with $P = \{A \longleftarrow A\}$, $I = \{A\}$, $D(P) = \emptyset$; furthermore I is also a fixed point).

The following definitions are only tools for the clarity of explanation.

Definition 3.4.

An incorrectness of P in I is an instance $A\theta \longleftarrow \mathcal{C}\theta$ of a clause $A \longleftarrow \mathcal{C}$ of P such that $\mathcal{C}\theta \subseteq I$ but $A\theta \notin I$. An insufficiency of P for I is an atom in $I - T_P(I)$.

It is obvious that

a) P is correct in I if and only if there is no incorrectness of P in I
b) P is sufficient for I if and only if there is no insufficiency of P for I.

The two logic programs coming from Shapiro [8] and studied here enable to diagnose an incorrectness or an insufficiency, under certain conditions. So, as Shapiro says, it is obvious that these programs can be applied only if the expected denotation is not a fixed point of T_P.

But, in order to be more precise, we add the following definitions :

Definition 3.5.

An incorrectness symptom of P in I is an atom in D(P)-I.
An insufficiency symptom of P for I is an atom in I-gfp (T_P).

Then, the two following obvious implications
$T_P(I) \subseteq I \implies D(P) \subseteq I$
$I \subseteq T_P(I) \implies I \subseteq gfp(T_P)$
are translated as

Proposition 3.6.

If there is an incorrectness (resp. insufficiency) symptom, then there is an incorrectness (resp.insufficiency).

But each of the two converses is false (already seen counter-examples : with $P = \{A \longleftarrow B\}$ and $I = \{B\}$, $A \longleftarrow B$ is an incorrectness without any symptom because $D(P) = \emptyset$;

with $P = \{A \longleftarrow B,\ B \longleftarrow\}$ and $I = \{A\}$, A is an insufficiency without any symptom because $gfp(T_P) \supseteq D(P) = \{A,B\} \supseteq I$; furthermore there is only one fixed point $D(P) = gfp(T_P)$).

The method to diagnose a bug (incorrectness or insufficiency) starts from a symptom. Then some bugs may escape from this diagnosis, and for applying the method it is necessary that I should not satify $D(P) \subseteq I \subseteq gfp(T_P)$. This condition is stronger than I not a fixed point of T_P.

3.2. Comments

The intuitive meaning of an incorrectness symptom A is clear : the program P gives an "answer" ($A \in D(P)$) but it is not consistent with what was expected ($A \notin I$). It is less clear for an insufficiency symptom, because $gfp(T_P)$ has no immediate intuitive meaning. But the complementary of $gfp(T_P)$ has a noteworthy subset : the "finite failure set" of P, that we shall note ffs(P) [1].

Roughly speaking, the idea is that starting from a "question" in ffs(P), the program P "fails" that is to say that the execution ends without giving any answer (this comment goes out of the non-procedural framework, and we shall return to this question further down (6.5)).

Then an atom A in $I \cap ffs(P)$ has the following intuitive meaning: it is an answer which ought to be confirmed ($A \in I$) but P does not confirm it ($A \in ffs(P)$). Now, such an atom is a particular case of an insufficiency symptom ($A \in ffs(P) \Longrightarrow A \notin gfp(T_P)$) and so the method applies in this case.

In order to consider that a logic program P_1 diagnoses a bug (incorrectness or insufficiency) in a logic program P, this bug must be represented by a term of an atom of the denotation of P_1. But the same P_1 must apply to different P and I, and then this denotation of P_1 must be relative (definition 2.22) to a set of atoms representing P and I in a way.

There are two "levels of language" : atoms and clauses at the level of P must be seen as terms at the level of P_1, where predicate symbols must represent the properties : being an insufficiency, an incorrectness, a clause of P, an atom valid in I...

4.Technical preliminaries

4.1. Representations

We introduce three new function symbols, that we write empty, and, if with respective arity 0,2,2, and we define a new function symbols system $\mathcal{F}_0$ By joining the symbols of $\mathcal{F}$, those of $\mathcal{P}$, and {empty, and, if}. The symbols of $\mathcal{P}$, with their arity as predicate symbols, then become function symbols of $\mathcal{F}_0$ so that an obvious identification gives TERM($\mathcal{F}$) $\subseteq$ TERM($\mathcal{F}_0$) and ATOM($\mathcal{P}$,$\mathcal{F}$) $\subseteq$ TERM($\mathcal{F}_0$).

By convention we shall write "t_1 and t_2" for "and(t_1, t_2)", "t_1 if t_2" for "if(t_1, t_2)" and if $n \geq 3$, "t_1 and t_2...and t_n" for "t_1 and (t_2...and t_n).

We shall call conjunctions the terms of TERM($\mathcal{F}_0$) of one of the three following forms :

a) empty,

b) A, for $A \in$ ATOM($\mathcal{P}$,$\mathcal{F}$)

c) A_1 and...A_n, for $n \geq 2$ and $A_i \in$ ATOM($\mathcal{P}$,$\mathcal{F}$).

We shall call implications the terms of the form A if Q where $A \in$ ATOM($\mathcal{P}$,$\mathcal{F}$) and Q is a conjunction.

We shall say that a conjunction represents a set of atoms defined as follows : empty, A, A_1 and...A_n respectively represent the sets $\emptyset$, $\{A\}$, $\{A_1,\ldots,A_n\}$.

We shall say that an implication A if Q represents the clause $A \longleftarrow \mathcal{C}$ where $\mathcal{C}$ is the set represented by Q; and that a set of implications represents a logic program : the set of the clauses represented by the implications.

If RP is any set of implications, then INST(RP) will be the set of all the instances of the elements of RP by substitutions with range in TERM($\mathcal{F}$).

So let P be the logic program represented by RP, and $I \subseteq ATOM(\mathcal{P},\mathcal{F})$. I is a model of P if and only if , for each implication A if Q $\in$ INST(RP), if the set represented by Q is enclosed in I, then A $\in$ I.

4.2. Ramifications

The tool for exactly describing the link between bug and symptom will be a forest, more precisely a (finite) sequence of (non-empty, finite) trees, where the information at a node is an atom of $ATOM(\mathcal{P},\mathcal{F})$.

We shall use Pair's formalism [6] which is handy for doing double inductions, and so speak of ramifications(on the set $ATOM(\mathcal{P},\mathcal{F})$) for such forests.

On the ramifications two operations are defined : the concatenation, noted +, which is associative and has as zero element the empty ramification noted Λ, and the taking root operation, noted $\times$, which to an atom A and a ramification r associates the ramification $A\times r$: $A\times r$ may be seen as a tree having A at the root, the sequence of sub-trees under the root being r.

The ramification $A\times\Lambda$ is identified with the atom A, and each non-empty ramification r is written in a unique manner $r = A\times r_1+r_2$, where A is an atom and r_1 and r_2 are ramifications ($\times$ has priority on +).

What we call roots of a ramification are exactly atoms : the set of the roots of Λ is the empty set, the one of $A\times r_1+r_2$ is the union of $\{A\}$

with the one of r_2. In a similar way we speak of the set of all the atoms, and of the set of the leaves of a ramification.

A tree is a ramification of the form $A*r$, and the sub-trees of a ramification are trees defined as follows : the set of the sub-trees of $\wedge$ is the empty set, the one of $A*r_1+r_2$ is the union of $\{A*r_1\}$ with the one of r_1 and the one of r_2.

The root conjunction of a ramification is defined as follows : the root conjunction of $\wedge$ is the term empty, the one of $A*r_1$ is A, and, if $r_2 \neq \wedge$, the one of $A*r_1+r_2$ is A and Q where Q is the one of r_2.

The root implication of the tree $A*r$ is A if Q where Q is the root conjunction of r.

Definition 4.1

The ramification r is conformable to the set of implications RP if and only if, for each sub-tree of r, the root implication is in INST(RP).

Proposition 4.2.

Let P be the logic program represented by the set of implications RP. Then each ramification conformable to RP has all its roots in D(P) (all the roots of the sub-trees as well). Conversely, each atom in D(P) is the root of a tree conformable to RP.

proof : firstly by induction on the ramifications ; secondly by induction on D(P) that is to say by using that D(P) is enclosed in every I such that $T_P(I) \subseteq I$.

5. The problem of non-correctness

5.1. Critical implication

Definition 5.1.

Given a set of implications RP and an expected denotation I, a tree A×r is RP-I-Critical if

a) A×r is conformable to RP

b) $A \notin I$

c) All the atoms of r are in I.

An RP-I-critical implication of a ramification r_1 is the root implication of a sub-tree of r_1 which is RP-I-critical.

It is easily seen that :

a) Let P be the logic program represented by RP. Then the clause represented by an RP-I-critical implication is an incorrectness of P in I (definition 3.4.).

b) If a ramification is conformable to RP, then it has a RP-I-critical implication if and only if it has an atom which is not in I (see such a minimal, in an obvious sense, atom).

5.2. The logic program "incorrectness diagnoser"

We add to $\mathcal{F}_0$ a new constant symbol, noted not-defined and so we have a new function symbols system $\mathcal{F}_1$.

Now let $\mathcal{P}_1$ be the predicate symbols system made of the symbols incorrectness, clause, valid, unsatisfiable with respective arity 2,1,1,1.

Now it is with ATOM($\mathcal{P}_1$, $\mathcal{F}_1$) that we define new clauses, and in particular we consider the following logic program. This program is inspired by Shapiro's program [8] for non-correctness, but it is adapted to the theoretical framework of this paper (extended in a way).

Further down (5.5) we shall see another program closer to Shapiro's one. The logic program "incorrectness diagnoser" is the set of the 7 following clauses :

incorrectness(empty, not-defined) <—

incorrectness(x and y, u if v) <— incorrectness(x,u if v)

incorrectness(x and y, u if v) <— incorrectness(y,u if v)

incorrectness(x and y, not-defined) <— incorrectness(x,not-defined), incorrectness(y, not-defined)

incorrectness(x,u if v) <— clause(x if y), incorrectness(y,u if v)

incorrectness(x, not-defined) <— clause(x if y), incorrectness(y,not-defined), valid(x)

incorrectness(x,x if y) <— clause(x if y), incorrectness(y, not-defined), unsatisfiable(x)

The intention is the following : Let I be an expected denotation $\subseteq$ Atom($\mathcal{P}$, $\mathcal{F}$) and P a logic program whose clauses are defined with atoms in ATOM($\mathcal{P}$,$\mathcal{F}$). Let RP be a set of implications representing P (so RP $\subseteq$ TERM ($\mathcal{F}_1$)). Let A be an incorrectness symptom of P in I (definition 3.5.). Then A is the root of a tree conformable to RP (because A $\in$ D(P)) and that tree has a RP-I-critical implication (because A $\notin$ I). Let us note it B if Q, it represents an incorrectness of P in I, that we may consider as a "cause" of the symptom A.

We would like the logic program "incorrectness diagnoser" to detect this bug in the sense that the atom incorrectness(A, B if Q) should be in its denotation (now a subset of ATOM ($\mathcal{P}_1$, $\mathcal{F}_1$)).

It may only be a matter of relative denotation, which depends on P and I, therefore relative to a set $\mathcal{E} \subseteq$ ATOM($\mathcal{P}_1$, $\mathcal{F}_1$). In $\mathcal{E}$ there will be the atoms clause (A if Q) for the A if Q $\in$ RP, and moreover the atoms valid (A) and unsatisfiable (B) for the A, B $\in$ ATOM($\mathcal{P}$,$\mathcal{F}$) with A valid in I, and B unsatisfiable in I, in a sense to come.

The predicate clause is the parallel of the predicate used by Shapiro, in another framework, for having access to the clauses of the erroneous program; the predicates valid and unsatisfiable are the parallels of the predicates used by Shapiro for asking questions to the "oracle" about the semantics of this program. (in practice, the role of "oracle" is played by the programmer or a base of already recorded answers).

For recursivity reasons, the suitable terms are not only trees and roots but ramifications and root conjunctions.

5.3. Properties of the program "incorrectness diagnoser"

We are going to study the two problems which naturally arise in order to apply this program. (Figuratively, suppose that we have found an incorrectness symptom A and that we "ask the question" incorrectness(A,x).

1st problem :

> If incorrectness (A, B if Q) is in the denotation of this program, is it true that B if Q represents an incorrection of P in I, and secondarily that B if Q is a RP-I-critical implication of a tree with root A ?

2nd problem :

> If A is an incorrectness symptom of P in I, and if B if Q is a critical implication of a tree conformable to RP and having A as root, is it true that incorrectness (A, B if Q) is in the denotation of this program ?

These two problems are the most useful part of the problem of total correctness of the program "incorrectness diagnoser". To deal with this problem completely would be a tedious task, in particular the complete definition of its expected denotation ! (On the one hand it must be a relative denotation, on the other hand there are "pathological" atoms coming from mixing $\mathcal{F}$, $\mathcal{P}$ and {empty, and, if}). However the following proofs show how we could tackle the problem. Technically it would be easier to extend the notion of conjunction...

The first problem can be seen as a part of a partial correctness problem, and the second one as a part of a completeness problem (problems not at the same "level" as those of correctness and sufficiency of P).

We can hope a precise answer only after precising the set $\mathcal{E}$, relatively to which we take the denotation of the program "incorrectness diagnoser" (precising what is an atom unsatisfiable in I). But from now on we can state a "limitation" result (incompleteness in a way) :

Proposition 5.2.

No logic program with the set of terms TERM($\mathcal{F}_1$) can solve the two previous problems at the same time (that is to say, for any P and I, give positive answer).

This result shows that it is a limitation intrinsic in the problem, and not linked to the diagnosis method. Moreover the very simple argumentation shows that this result comes from the representation of atoms and clauses by terms and is independent from the particular formalism (TERM($\mathcal{F}_1$)) used for this representation.

proof : as counter-example, take P and I with a clause in P (the simpler one is R(x) <—) which is not valid in I, but which has a valid instance (for example R(1) <—), and remark that the denotation of the program "incorrectness diagnoser" must be closed by substitution (it must have incorrectness (R(x), R(x) if empty) - 2^{nd}problem- thus also incorrectness (R(1), R(1) if empty), but this is contradictory -1^{st}problem-).

However we can give a partial solution to these problems.

Definition 5.3.

Given an expected denotation $I \subseteq$ ATOM($\mathcal{P}$, $\mathcal{F}$), the atom A is unsatisfiable in I if, for any substitution θ, $A\theta \notin I$.

(In usual logic, in the ℓ-interpretation corresponding to I, A is false for any assignment θ).

Notation 5.4.

I being an expected denotation and RP a set of implications,
$\mathcal{E}(RP,I)$ = the set made of the atoms

a) clause(A if Q) for the A if Q $\in$ RP

b) valid(A) for the A valid in I

c) unsatisfiable(A) for the A unsatisfiable in I.

$\mathcal{D}(RP, I)$ = the denotation of the program "incorrectness diagnoser" relative to $\mathcal{E}(RP, I)$.

The following theorem gives a partial solution to the 2nd problem :

Theorem 5.5.

If Q_1 is the root conjuction of a ramification which is conformable to RP and has all its atoms in I, then incorrectness (Q_1, not-defined) $\in \mathcal{D}(RP,I)$. If Q_1 is the root conjonction of a ramification which is conformable to RP, and has a RP-I-critical implication B if Q with B unsatisfiable in I, then incorrectness (Q_1, B if Q) $\in \mathcal{D}(RP,I)$.

proof : a careful but easy checking by induction on ramifications.

We remark that if B is a closed atom, B unsatisfiable in I is equivalent to B $\notin$ I. So we may say that any closed incorrectness associated to an incorrectness symptom is diagnosed.

The following theorem now gives a total solution to the 1st problem :

Theorem 5.6.

If Q_1 is a conjunction such that
incorrectness(Q_1, not-defined) $\in \mathfrak{D}(RP,I)$ then Q_1 is the root conjunction of a ramification which is conformable to RP and has all its atoms in I.
If Q_1 is a conjuction and B if Q is an implication such that
incorrectness(Q_1, B if Q) $\in \mathfrak{D}(RP,I)$ then Q_1 is the root conjunction of a ramification (which is not necessarily conformable to RP) which has B if Q as a RP-I-critical implication.

proof : the basic idea is checking by induction on $\mathfrak{D}(RP,I)$ as for proposition 4.2. : $\mathfrak{D}(RP,I)$ is the (absolute) denotation of a program made of the 7 clauses of the program "incorrectness diagnoser" and of the clauses of $\mathfrak{E}(RP,I)$ (definition 2.22). $\mathfrak{E}(RP,I)$ already says what the atoms of $\mathfrak{D}(RP,I)$ of the form clause(...), valid(...), unsatisfiable(...) are.

But we do not attempt to entirely describe $\mathfrak{D}(RP,I)$. What we only prove by induction on atoms of $\mathfrak{D}(RP,I)$ is a property of the following form : either it is an atom clause(...), valid(...), unsatisfiable(...) or, if it is of the form incorrectness(Q_1,...) considered by the theorem (Q_1 is a conjunction,...) then this atom satisfies what the theorem says (there is a ramification such that...).

The idea is to inspect each of the 7 clauses, and to associate to it a concatenation of ramifications or a "taking root". More precisely, what the induction requires is a checking for each instance of each clause. But what prevents the proof from being only a careful routine checking is this notion of instance : a proof by induction on $\mathfrak{D}(RP,I)$ requires substitutions with range in TERM($\mathfrak{F}_1$), but in conjunctions, implications and ramifications the terms must be only in TERM($\mathfrak{F}$).

One must be careful about some "pathological" atoms in $\mathfrak{D}(RP,I)$ which could generate some atoms in contradiction to the theorem (actually only the 5^{th} and 6^{th} clauses require a careful inspection because of the variable y which does not appears on the left). The remainder of the proof is devoted to these tedious technical precautions.

a) Let x_0 be fixed $\in \mathcal{U}$. We define a map $t \longmapsto \tilde{t}$ from TERM($\mathcal{F}_1$) to TERM ($\mathcal{F}$) :

$\tilde{*} = x$ for $x \in \mathcal{U}$

$\widetilde{\underline{\text{empty}}} = \widetilde{\underline{\text{not-defined}}} = x_0$

$\widetilde{f(t_1,\ldots t_n)} = f(\tilde{t}_1,\ldots,\tilde{t}_n)$ for f symbol of $\mathcal{F}$ and $t_1,\ldots,t_n$ in TERM($\mathcal{F}_1$)

$\widetilde{R(t_1,\ldots,t_m)} = x_0$ for R symbol of $\mathcal{P}$ and $t_1,\ldots,t_m$ in TERM($\mathcal{F}_1$)

$\widetilde{\underline{\text{and}}(t_1,t_2)} = \widetilde{\underline{\text{if}}(t_1,t_2)} = x_0$ for t_1,t_2 in TERM($\mathcal{F}_1$).

We have $\tilde{t} = t$ if $t \in$ TERM($\mathcal{F}$).

b) For each $\theta : \mathcal{U} \longrightarrow$ TERM($\mathcal{F}_1$) we define
$\tilde{\theta} : \mathcal{U} \longrightarrow$ TERM($\mathcal{F}$) by $\tilde{\theta}(x) = \widetilde{\theta(x)}$.
We have $\widetilde{t\theta} = t\tilde{\theta}$ if $t \in$ TERM($\mathcal{F}$).

c) We call generalized atom a term in TERM($\mathcal{F}_1$) of the form $R(t_1,\ldots,t_m)$ with R symbol of $\mathcal{P}$, and we define

$\widehat{R(t_1,\ldots,t_m)} = R(\tilde{t}_1,\ldots,\tilde{t}_m) \in$ ATOM($\mathcal{P},\mathcal{F}$).

We have $\hat{A} = A$ if $A \in$ ATOM($\mathcal{P}$, $\mathcal{F}$). In the same way we define a generalized conjunction and a generalized implication (in conjunction and implication we replace atom of ATOM($\mathcal{P},\mathcal{F}$) by generalized atom) and we extend $\hat{}$ by :

$\widehat{\underline{\text{empty}}} = \underline{\text{empty}}$, $\widehat{A_1 \ \underline{\text{and}} \ A_2} = \hat{A}_1 \ \underline{\text{and}} \ \hat{A}_2$, $\widehat{A \ \underline{\text{if}} \ Q} = \hat{A} \ \underline{\text{if}} \ \hat{Q}$.

d) For any $A \in ATOM(\mathcal{P}, \mathcal{F})$ and $\theta : \mathcal{V} \longrightarrow TERM(\mathcal{F}_1)$ $A\theta$ is a generalized atom, and $\widehat{A\theta} = A\tilde{\theta}$. It is the same for a conjunction and an implication.

e) If the atom valid(t) is in $\mathcal{D}(RP,I)$ then $t = A\theta$ with $A \in ATOM(\mathcal{P},\mathcal{F})$, A valid in I and $\theta : \mathcal{V} \longrightarrow TERM(\mathcal{F}_1)$. Thus t is a generalized atom and $\hat{t} = A\tilde{\theta}$ is valid in I. It is the same for unsatisfiable(t) and unsatisfiable in I.

f) If the atom clause $(t_1$ if $t_2)$ is in $\mathcal{D}(RP,I)$ then t_1 if $t_2 = (A$ if $Q)\theta$ with A if $Q \in RP$ and $\theta : \mathcal{V} \longrightarrow TERM(\mathcal{F}_1)$. Thus t_1 if t_2 is a generalized implication and

$\widehat{t_1 \text{ if } t_2} = (A \text{ if } Q)\tilde{\theta} \in INST(RP)$.

g) Now, it remains to check, by induction, a stronger property than in the theorem : If Q_1 is a generalized conjunction such that incorrectness $(Q_1$, not-defined$) \in \mathcal{D}(RP,I)$ then $\hat{Q}_1$ is the root conjunction of a ramification which is conformable to RP and has all its atoms in I; and it is the same for the remainder of the theorem with Q_1 and B if Q generalized.

5.4. Extension to relative correctness

The correctness problem of P has been formulated concerning the absolute denotation of P. But we can extend it by now considering the denotation of P relative to some set of atoms $\mathcal{A}$. This extension is parallel to the fact that Shapiro, in another framework, admits in the erroneous (PROLOG) program some "system predicates" such as for example arithmetical predicates.

It is possible to restart with more general new definitions and give more general new proofs. In order to avoid it, on the contrary we are going to use previous results by using the definition of the denotation of P relative to $\mathcal{A}$ as the (absolute) denotation of $P \cup \dot{\mathcal{A}}$. We make the definitions of correct, incorrectness, incorrectness symptom (definitions 3.3, 3.4, 3.5) relative to $\mathcal{A}$ by replacing P in the previous definitions by $P \cup \dot{\mathcal{A}}$. But, in view of the motivation, the interesting case is when the correctness problems are confined in P, and do not concern $\mathcal{A}$, formally the case where $\mathcal{A} \subseteq I$.

In this case, it is easily seen that

a) an incorrectness symptom of P in I relative to $\mathcal{A}$ never is an instance of an atom in $\mathcal{A}$.

b) an incorrectness of P in I relative to $\mathcal{A}$ necessarily is an instance of a clause in P.

c) P is correct in I relatively to $\mathcal{A}$ if and only if I is a model of P.

Let us again consider a set of implications RP representing P. Let us also note $R\mathcal{A}$ the set of the implications A *if empty* where $A \in \mathcal{A}$; $R\mathcal{A}$ represents the set of clauses $\dot{\mathcal{A}}$.

Of course we can apply the previous results by replacing P by $P \cup \dot{\mathcal{A}}$ and RP by $RP \cup R\mathcal{A}$, but then the denotation $\mathcal{D}(RP \cup R\mathcal{A}, I)$ of the program "incorrectness diagnoser" is relative to $\mathcal{E}(RP \cup R\mathcal{A}, I)$ (notation 5.4.).

Now, this set $\mathcal{E}(RP \cup R\mathcal{A}, I)$ is inadequate as concerns the motivation, because it represents $A \in \mathcal{A}$ in a way which is not natural : by the trick of *clause*(A *if empty*), as the clause A ⟵. Now there is a parallel with the fact that, in another framework in PROLOG, the "system predicates" which recognize the clauses and which recognize the "system predicates" are not the same.

In other terms, the results we want about the diagnosis of relative incorrectness are already in the denotation of the program "incorrectness diagnoser" relative to $\mathcal{E}(RP \cup R\mathcal{A}, I)$ but we want another set in place of $\mathcal{E}(RP \cup R\mathcal{A}, I)$.

One adds to $\mathcal{P}_1$ a new predicate symbol with arity 1, noted $\underline{a}$.

We call "extended incorrectness diagnoser" the logic program obtained by adding to the program "incorrectness diagnoser" the following 8^{th} clause :

incorrectness(x,not-defined) <— $\underline{a}$(x)

Then the theorems 5.5. and 5.6. remain true when we consider this program, but now its denotation relative to the set $\mathcal{E}(RP,I) \cup \{\underline{a}(A) \mid A \in \mathcal{A}\}$. (note that clause(A if empty) with $A \in \mathcal{A}$ could contribute to the denotation of the former program relative to $\mathcal{E}(RP \cup R\mathcal{A}, I)$ only thanks to its 6^{th} clause (because $\mathcal{A} \subseteq I$), and that contribution is replaced by the one of $\underline{a}(A)$ thanks to the 8^{th} clause of the new program).

5.5. Comparison with Shapiro's program

If, keeping our notations we exactly transpose Shapiro's program [8] for incorrectness (in another framework), we obtain a new program : the 3^{th} and 4^{th} clauses of our program are replaced by only the following clause :

incorrectness(x and y,z) <— incorrectness(x,not-defined),
incorrectness(y,z)

It is easy to see that the denotation of this new program (relative to $\mathcal{E}(RP,I)$) is a part of the one of our program. More precisely, the theorems 5.5. and 5.6. remain true for this new programm provided that we replace "RP-I-critical implication" by "first RP-I-critical implication", where "first" refers to a "post-order"

that we can easily define on the sub-trees of a ramification. This restriction is natural in the procedural framework of Shapiro, and corresponds to the search strategy of the interpreter, but it is less meaningful in the framework of our paper.

We shall see that the same comparison question, for the problem of insufficiency, requires deeper comments.

6. The problem of non-sufficiency

In view of the definitions, this problem is, in a way, dual of the previous problem of non-correctness. We are going to use a similar method and to go less through details.

6.1. Quasi-conformable ramification

We say that an atom B is a sub-root of a ramification r if there is a sub-tree A r_1 of r such that B is an atom of r_1.

Definition 6.1.

Let RP be a set of implications, I an expected denotation and r a ramification. Then r is quasi-conformable to RP and I if
a) All the sub-roots of r are in I
b) For each sub-tree A×r_1 of r such that $r_1 \neq \wedge$, the root implication of A×r_1 is in INST(RP).

lemme 6.2.

Let P be the logic program represented by RP, and A an atom not in $gfp(T_P)$. Then there is a tree whose root is A, which is quasi-conformable to RP and I, and which has at least one leaf not in $T_P(I)$.

proof : we suppose that all the trees whose root is A and which are quasi-conformable to RP and I have all their leaves in $T_P(I)$, and we prove that then $A \in gfp(T_P)$.

With the following argumentation we avoid a formalism too heavy :

We define r_0 = A, it is a tree with root A, quasi-conformable to RP and I. A is a leaf of r_0, thus $A \notin T_P(I)$, thus there is A if Q in INST(RP) with all the atoms of Q in I. By an obvious "taking root" of

these atoms of Q with A, we obtain a tree r_1 with root A, and again quasi-conformable to RP and I (the case Q = empty is not excluded, then $r_1 = r_0 = A$ and already $A \in D(P) \subseteq gfp(T_P)$).

So by induction we define a sequence $r_0,\ldots,r_n,\ldots$ of trees all with root A and quasi-conformable to RP and I, r_{n+1} being obtained by possibly hanging up new atoms of I to the leaves of r_n (all in $T_P(I)$ by hypothesis), thanks to implications in INST(RP) (if we hang up no atom any longer, then $r_n = r_{n+1} \ldots$ is a tree conformable to RP, and already $A \in D(P) \subseteq gfp(T_P)$).

$r_0,\ldots,r_n,\ldots$ can be seen as successive approximations of a tree possibly infinite.

Let J be the set of all the atoms of all the r_n. Then by construction $A \in J \subseteq T_P(J)$, thus $J \subseteq gfp(T_P)$, thus $A \in gfp(T_P)$.

6.2. The logic program "insufficiency diagnoser"

Now we add to $\mathcal{F}_0$ not only the constant symbol not-defined, but a new function symbol, noted atom, with arity 1, and we obtain a new function symbols system, noted $\mathcal{F}_2$.

Now let $\mathcal{P}_2$ be the predicate symbols system made of the symbols insufficiency, clause, satisfiable, impossible with respctive arities 2, 1, 1, 1.

The logic program "insuffiency diagnoser" is the set of the 8 following clauses :

```
insufficiency(empty, not-defined)   <—

insufficiency(x and y, atom(u))     <— insufficiency(x, atom(u))

insufficiency(x and y, atom(u))     <— insufficiency(y, atom(u))

insufficiency(x and y, not-defined) <— insufficiency(x, not-defined)),
                                       insufficiency(y, not-defined)

insufficiency(x,z)                  <— clause(x if y),
                                       satisfiable(y),
                                       insufficiency(y,z)

insufficiency(x, atom(x))           <— impossible(x)

satisfiable(empty)                  <—

satisfiable(x and y)                <— satisfiable(x),
                                       satisfiable(y)
```

This logic program is inspired by Shapiro's program [8] for non-sufficiency, but it is adapted to the theoretical framework of this paper. Furtherdown (6.5) we return to the comparison of the two programs.

The intention is the following :

Let A be an insufficiency symptom of P for I. A is root of a tree, quasi-conformable to RP and I, and which has at least one leaf B not in $T_P(I)$ (because $A \notin gfp(T_P)$). But all the atoms of this tree are in I (even $A \in I$) thus we are sure that $B \in I$, and that B is an insufficiency of P for I (definition 3.4), that we consider as a "cause" of the symptom A.

We would like the atom insufficiency(A,atom(B)) to be in the denotation of this program (the symbol atom is a formal trick (Shapiro [8]) which plays the role played by if in incorrectness diagnosis, which was to distinguish between an implication and the constant not-defined).

This denotation is relative to a set $\mathcal{E}'$ which again contains the atoms clause(A if Q) for the A if Q $\in$ RP, and the atoms satisfiable(A) for the A valid in I. $\mathcal{E}'$ also contains atoms impossible(A) for some A (to precise) which are insufficiencies of P for I.

In the parallel with Shapiro's framework, we notice that for what corresponds to impossible(x), Shapiro uses the "control structures" not [8] or if-then-else [9], joined with the predicates giving access to the clauses and asking the oracle. Here, one asks the oracle to give a valid instance of a given atom.

6.3. Properties of the program "insufficiency diagnoser"

Again naturally two problems arise :

3th problem :

> If A $\in$ I and if insufficiency(A, atom(B)) is in the denotation of this program, is it true that B is an insufficiency of P for I, and secondarily that B is a leaf of a tree with root A quasi-conformable to RP and I ?

4th problem :

> If A is an insufficiency symptom of P for I and if B is an insufficiency of P for I which is a leaf of a tree with root A quasi-conformable to RP

and I, is it true that insufficiency(A, atom(B)) is in the denotation of this program ?

Again the 3rd problem can be seen as a part of a partial correctness problem, and the 4th as a part of a completeness (not sufficiency!) problem.

Again we have the same limitation or incompleteness result as previously, which makes it impossible to solve the two problems at the same time. The argumentation is similar : use an insufficiency B an instance Bθ of which is not an insufficiency (the simpler counter-example is with $I = \{R(t) \mid t \in TERM(\mathcal{F})\}$, $P = \{R(1) \leftarrow\}$ then R(x) is an insufficiency, but not R(1). Furthermore here $gfp(T_P) = \{R(1)\}$ and R(x) is an insufficiency symptom of P for I).

Definition 6.3

An atom A is an impossibility of P for I if, for each subtitution θ, $A\theta \notin T_P(I)$.

(we note in particular that $A \notin T_P(I)$, and if $A \in I$ then A is an insufficiency of P for I).

Notations 6.4.

$\mathcal{E}'(RP,I)$ = the set made of the atoms

a) clause(A if Q) for the A if Q ∈ RP

b) satisfiable(A) for the A valid in I

c) impossible(A) for the A which are impossibilities of P for I.

$\mathcal{D}'(RP,I)$ = the denotation of the program "insufficiency diagnoser" relative to $\mathcal{E}'(RP,I)$.

From the following theorem we deduce a partial solution to the 4th problem :

Theorem 6.5.

If Q is the root conjunction of a ramification which is conformable to RP and has all its sub-roots in I, then
insufficiency(Q, not-defined) $\in \mathfrak{D}'(RP,I)$

If Q is the root conjunction of a ramification which is quasi-conformable to RP and I and has a leaf B which is an impossibility of P for I, then insufficiency(Q, atom (B)) $\in \mathfrak{D}'(RP,I)$.

proof : as theorem 5.5.

We notice that if B is closed, B impossibility of P for I is equivalent to $B \notin T_P(I)$. So we may say that any closed insufficiency associated to an insufficiency symptom is diagnosed.

From the following theorem we deduce a total solution to the 3rd problem :

Theorem 6.6.

If Q is a conjunction such that
insufficiency(Q, not-defined) $\in \mathfrak{D}'(RP,I)$ then Q is the root conjunction of a ramification which is conformable to RP and has all its sub-roots in I.

If Q is a conjunction and B an atom such that
insufficiency(Q, atom(B)) $\in \mathfrak{D}'(RP,I)$ then Q is the root conjunction of a ramification which is quasi-conformable to RP and I, and has B as leaf with $B \notin T_P(I)$.

proof : as theorem 5.6.

6.4. Extension to relative sufficiency

As for incorrectness (5.4) we can make the definitions about sufficiency relative to $\mathfrak{A} \subseteq \text{ATOM}(\mathcal{P}, \mathcal{F})$.

We notice that $T_{P \cup \overleftarrow{\mathfrak{A}}}(I) = T_P(I) \cup \text{INST}(\mathfrak{A})$ (where $\text{INST}(\mathfrak{A})$ is the set of all the instances of $\mathfrak{A}$), thus each fixed point of $T_{P \cup \overleftarrow{\mathfrak{A}}}$, and in particular $\text{gfp}(T_{P \cup \overleftarrow{\mathfrak{A}}})$ contains $\text{INST}(\mathfrak{A})$. So an insufficiency, and an insufficiency symptom, of P for I, relative to $\mathfrak{A}$, never are instances of atoms in $\mathfrak{A}$.

As for incorrectness, we call "extended insufficiency diagnoser" the logic program obtained by adding to the program "insufficiency diagnoser" the following 9^{th} clause :

insufficiency(x, not-defined) <— a(x)

Then the theorems 6.5. and 6.6. remain true with this new program, but with its denotation relative to $\mathcal{E}'(RP,I) \cup \{\underline{a}(A) \mid A \in \mathfrak{A}\}$.

6.5. Comparison with Shapiro's program and "finite failure"

If, keeping our notations, we exactly transpose Shapiro's program [8] for insufficiency (in another framework), we obtain a new program : the 3^{rd} and 4^{th} clauses of our program are replaced by only the following clause :

insufficiency(x and y,z) <— insufficiency(x, not-defined), insufficiency(y,z)

It is easy to see that the denotation of this new program (relative to $\mathcal{E}(RP,I)$) is a part of the one of our program : in practice we can replace an application of the 3^{rd} clause of this new program by

an application of the 3^{rd} clause of our program if z = atom (u) or of the 4^{th} clause of our program if z = not-defined. Then theorem 6.6. can again be applied, that is to say that the 3^{rd} problem is solved for this new program (problem which can be seen as the most useful part of the problem of partial correctness of this new program).

It is not the same for the 4^{th} problem (part of a completeness problem) even if we limit the problem to the closed denotation.

However Shapiro in his framework shows that his program solves the given problem :

In fact in pratice if the programmer attempts to diagnose an insufficiency, starting from an insufficiency symptom A, it is because the experiment has proved that A makes the program P "finitely fail" (see 3.2.). In practice it is sufficient that insufficiency diagnosis applies to this type of symptom, and it is so in Shapiro's framework.

Let us see what can be said in the framework of "fixpoint", non-procedural semantics.

It is natural to think about the "finite failure set", already mentioned (in 3.2.).

In a way $gfp(T_P)$ is dual of D(P) and an insufficiency symptom ($A \in I\text{-}gfp(T_P)$) is dual of an incorrectness symptom ($A \in D(P)\text{-}I$). But this duality is not complete : if we write $DD(P) = \bigcap_{n \in \mathbb{N}} T_P^n (ATOM(\mathcal{P}, \mathcal{F}))$ then DD(P) also is dual of D(P), in another way (2.5).

It is easy to see that each fixed point is included in DD(P), thus $gfp(T_P) \subseteq DD(P)$. The equality is not always true, it is true if we continue the definition of T_P^n with transfinite ordinal $\mathfrak{n}$ [1]. Then it is natural to say that A is a finite insufficiency symptom of P for I if $A \in I\text{-}DD(P)$, which is stronger than insufficiency symptom ($A \in I\text{-}gfp(T_P)$).

Moreover Apt and Van Emden [1] have proved that the finite failure set (3.2.) ffs(P) is equal to $ATOM(\mathcal{P}, \mathcal{F}) - DD(P)$. More exactly they consider only $ATOMC(\mathcal{P}, \mathcal{F})$ but the following remarks remain valid with this set of atoms.

The program "insufficiency diagnoser" in particular applies to a finite insufficiency symptom. But it is not true for this new program transposed from Shapiro [8] any longer.

The simpler counter-example is the following :

$I = \{A,B,C\}$, $P = \{A \longleftarrow B,\ C\ ;\ B \longleftarrow B\}$

then $D(P) = \emptyset$, $gfp(T_P) = DD(P) = \{B\}$, A is a finite insufficiency symptom, $A \in ffs(P)$, the insufficiency to detect is C.

If we represent P by

RP = {A *if* B *and* C, B *if* B } then we easily see that the atom *insufficiency*(A,C) is not in the denotation of this new program (relative to $\mathcal{E}'(RP,I)$). On the contrary if we represent P by

RP = {A *if* C *and* B, B *if* B }, it is in this denotation.

So the results are differents according to the *representation* of P, but it is not a drawback of the formalism used here, rather we must consider that it is a formal expression of the fact that the properties we are discussing now are not intrinsic in a logic program :

In fact, in Shapiro's framework his program solves the given problem because his notion of *finite failure* depends on the search strategy of the interpreter, which itself depends on the *writing* of the program. In other terms, the notion of finite failure in Shapiro's framework is not the notion of finite failure in the framework of logic programming characterized by Apt and Van Emden, which is the framework of this paper.

7. Conclusion

In this conclusion, we quit the theoretical framework of this paper and we allude to the perspective of the implementation in PROLOG of bug diagnosis programs for PROLOG programs. Shapiro [9] explains the reasons why it is desirable to use PROLOG on the two "levels". We do not touch on this question and we content ourselves with some remarks on the application of this work.

On the one hand we can consider that we have in a way described a theoretical "envelope" that an implementation must fill for the best, considering the difference between the theoretical interpreter "SLD resolution" and a PROLOG interpreter (in particular the question of finite failure). This prospect may be convenient when the "erroneous" programs are in "pure PROLOG", or more generally when they have predicates whose computation may be represented with the notion of relative denotation. These predicats remain to be studied (arithmetical predicates, particular use of input-output predicates...). According to this prospect, we apply theoretical results by writing procedures for clause(x if y), valid(x),...

On the other hand, we must extend bug diagnosis to programs with all PROLOG possibilities, including control predicates. (Shapiro [9] examines this question from a pragmatic point of view). But this leads beyond the theoretical framework of this paper.

Acknowledgements

I insist on expressing my profound gratitude towards Pierre Deransart who, at INRIA, has introduced me to logic programming, has given me this research subject and has advised and encouraged me.

I also insist on thanking Bernard Lorho for his support, as well as the members of the group "Langages et Traducteurs".

References.

[1] K.R. APT and M.H. VAN EMDEN,
Contributions to the Theory of Logic Programming, Journal of the ACM 29(3) : 841 - 863, July, 1982.

[2] K.L. CLARK,
Predicate Logic as a computational formalism, Res. Rep., Dep. of Computing, Imperial College, London 1979.

[3] M.H. VAN EMDEN and R.A. KOWALSKI,
The Semantic of predicate logic as a programming language, Journal of the ACM 23(4) : 733 - 742, October 1976

[4] R.A. KOWALSKI,
Logic programming, Proc. IFIP 1983

[5] P.PADAWITZ,
Resolution in Equational Theories : How Algebraic and Logic Programming Fit Together, April 1983, private communication to P. DERANSART.

[6] C. PAIR,
Théorie des Langages, EDF - CEA - IRIA, Septembre 1973

[7] H. ROGERS, Jr.
Theory of Recursive Functions and Effective Computability, Mc Graw-Hill, 1967.

[8] E.Y. SHAPIRO,
Algorithmic Program Debugging, Proc. 9th an. ACM Symp. on Principles of Programming Languages, 1982

[9] E.Y. SHAPIRO,
Algorithmic Program Debugging, MIT Press, 1983

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique